1 (2 puntos) OpenMP permite especificar distintos esquemas de planificación con la clausula `schedule`. Las opciones admitidas son `static`, `dynamic` y `guided`, ¿cuál se espera en general que tenga mejor rendimiento y por qué? En relación con esta cláusula ¿a qué se denomina *chunk* y para qué se utiliza?

SOLUCIÓN

La opción de planificación que debe comportarse mejor en un rango mayor de situaciones es `guided`, ya que intenta aprovechar las ventajas de las otras dos. A la hora de planificar las iteraciones de un bucle, comienza seleccionando bloques grandes de iteraciones, de modo que evita estar constantemente planificando (como podría ocurrir al emplear la opción `dynamic`). Por otro lado, `guided` evita los desequilibrios de carga presentes en `static`, ya que según se va avanzando en el bucle los bloques de iteraciones son más pequeños, con lo que se evita que haya procesadores ociosos mientras otros mantienen carga de trabajo.

Se denomina *chunk* a un parámetro opcional de la cláusula `schedule` que establece el número mínimo de iteraciones que se asignará a cada bloque (de iteraciones) a la hora de planificar.

2 (2 puntos) Explique el funcionamiento de la instrucción atómica `lock.xchg .Ri, /dir` e implemente para máquinas de memoria compartida un cerrojo con espera activa (*spin-lock*) empleando dicha instrucción.

SOLUCIÓN

Esta instrucción (*exchange* con el prefijo lock) está presente en algunas arquitecturas para que se puedan construir mecanismos de sincronización de memoria compartida de mayor abstracción, como pueden ser por ejemplo los cerrojos (*locks*) y las barreras (*barriers*).

Su ejecución con los operandos propuestos intercambia el contenido del registro Ri por el de la posición de memoria /dir de manera *atómica* o sin que ningún otro procesador pueda acceder a esa dirección hasta que termine de ejecutarse.

Una posible implementación de una primitiva para ganar, con la llamada espera activa, el acceso a un cerrojo podría ser, simplemente, tratar de reproducir el comportamiento de una instrucción de `test&set`, tal y como se desarrolló en una de las sesiones de ejercicios de la asignatura. Por simplicidad, podemos suponer que la ejecución de la instrucción afecta a los flags aritméticos (según el valor que retorna el registro Ri, en nuestro caso mientras no devuelva el valor cero) y, también, que se trata de una arquitectura genérica que permita los direccionamientos que empleamos en el código.

```
          LD .R0, #D'01
intento:  lock.xchg .R0, /dir_cerrojo
          bnz $intento
          ret
```

donde se debe notar que no haría falta volver a llevar un uno al registro empleado en cada vuelta (o *spin*) (puesto que devuelve ese valor en el registro R0) y que una implementación más eficiente sería la que hiciese primero la comprobación en lectura de que el cerrojo está abierto y luego intentase realizar su cierre en acceso atómico, lo que denominamos *test-and-test-and-set*.

3 (3 puntos) Para la implementación de una primitiva de LOCK(cerrojo) en un MP UMA con cachés privadas y un mecanismo de coherencia de cachés por *actualización* se consideran las tres opciones siguientes, A, B y C:

```
Implementación A

lock:
     while (LOAD(cerrojo) == 1) do
                nothing;
     STORE (1, cerrojo); almacena un '1' en la dirección del cerrojo

Implementación B

lock:
     while (TEST&SET (cerrojo) == 1) do
                nothing;

Implementación C

lock:
     repeat
         while (LOAD(cerrojo) == 1) do
                nothing;
     until (TEST&SET(cerrojo) == 0)
```

a) Explique brevemente el funcionamiento de las tres implementaciones. ¿Son todas implementaciones correctas de la primitiva LOCK para el sistema considerado? Indique razonadamente por qué.

b) Para cada una de las implementaciones que considere que funcionan correctamente, explique en qué medida es eficiente desde un punto de vista del rendimiento y por qué.

SOLUCIÓN

a) Si se analizan en un primer vistazo las tres soluciones, se puede observar que en todos los casos tienen una estructura de *spin-lock*. En **A** se consulta con `load` el valor del cerrojo hasta que sea 0 y entonces se escribe 1 en la dirección correspondiente con `store`. El mecanismo de coherencia haría que esta escritura se hiciese visible en el resto de las cachés que tuviesen copia del bloque del cerrojo. Sin embargo, desde la lectura hasta la escritura no existe la garantía de que se realice sin que pueda mediar la escritura de otro procesador y, por tanto, se trata de una implementación incorrecta. En **B** el `test&set` devuelve el valor previo de la dirección y escribe indiscriminadamente un uno en ella, y con la garantía de que estas dos acciones se realizan de manera atómica, a diferencia del caso anterior y, en consecuencia, sí se trata de una realización correcta. Por último, en **C** se realiza primero la comprobación de que el cerrojo está abierto y sólo cuando así sea se ejecuta un `test&set` para tratar de ganar su acceso, en un comportamiento que es entonces análogo al caso **B** (aunque sólo para los procesadores que hubieran pasado la comprobación primera) y por tanto funcionalmente correcto.

b) Sólo consideraremos aquí las implementaciones correctas, o sea, B y C. Con *actualización* como política de coherencia de cachés todas las copias de un mismo bloque se mantienen siempre actualizadas. En el caso de **B**, todos los procesadores detenidos en el lock darán lugar a que se produzca tráfico de actualización en el sistema debido a las escrituras que se propagan. Nótese, sin embargo, que no se producirían fallos de caché adicionales a los que ocurrirían en un comportamiento "monoprocesador": si un bloque se encuentra en caché será siempre copia válida. En **C** los procesadores en el lock no producirán fallos de caché en la primera fase (*spin* en la lectura del cerrojo) y, por tanto, ahora se reduce significativamente el tráfico extra debido a la ausencia de actualizaciones.

También es interesante destacar aquí que el patrón de *sharing* o "compartición" que se presupone en el acceso a un cerrojo es habitualmente de "grano fino", independientemente de que se haya reducido el número de escrituras. En el caso de una potencial "compartición secuencial", la actualización no parece la política de coherencia adecuada, por el exceso de tráfico que añade innecesariamente en el sistema de interconexión.

4 (1 punto) Indique la repercusión de los procesadores *multicore* y de las tarjetas gráficas (GPUs) en el mundo de la computación de altas prestaciones (HPC).

SOLUCIÓN

Estas dos tecnologías están teniendo un impacto fundamental en el campo del HPC. Hoy en día la práctica totalidad de los sistemas en el ránking Top500 tienen una estructura *multicore* (en definitiva, un multiprocesador en un chip), y, en su momento, hace un par de años, fue una innovación que contribuyó fundamentalmente a batir la barrera de los PFLOPS en computación de alto rendimiento.

El uso de los procesadores gráficos (GPU) como hardware específico para realizar operaciones masivas en coma flotante ha representado también un cambio cuantitativo fundamental en HPC, tanto es así que tres de las cinco primeras máquinas en el Top500 (el llamado "Top5"), en su última edición, emplean este tipo de procesadores. Aunque presentan cierta dificultad de programación, librerías como CUDA permiten que sin demasiado esfuerzo se le saque partido en computación científica a este tipo de procesadores. Su virtud fundamental es obtener rendimientos del orden TFLOPS a un coste muy reducido, algo impensable hasta hace muy poco.

5 (2 punto) Describa los estados en que puede estar un bloque de memoria caché en el protocolo de coherencia de cachés basado en invalidación llamado MESI.

SOLUCIÓN

Podríamos resumir los estados del modo siguiente. Hay dos estados, digamos, "extremos", el estado I (Invalid), en que el bloque no es válido y el estado M (Modified), en que es la única copia válida en toda la máquina (y que en su momento dará pie a que se actualice en memoria principal y/o a servir el fallo por invalidación que se dé en otro procesador). Los otros dos son el estado E (Exclusive), que representa una especie de paso previo al estado M (el bloque es también la única copia en caché, pero es coherente con la copia de memoria, que por lo tanto no necesitaría ser actualizada en *copy-back*), y el S (Shared), en que el bloque está potencialmente copiado en otras cachés y también es coherente con la copia de memoria. En este estado las escrituras se deberán difundir por el bus para que esas otras copias se invaliden.

1 (1'5 puntos) Indique las características fundamentales de los sistemas tipo UMA, NUMA y cluster. Comente brevemente por qué parece que la tecnología cluster y, más recientemente, la generalización del uso de procesadores gráficos (GPUs) se han acabado imponiendo en el mundo de HPC (computación de alto rendimiento).

SOLUCIÓN

Los dos primeros son máquinas con una visión de memoria compartida. En el caso UMA la memoria está también físicamente compartida y el tiempo de acceso es siempre el mismo. El sistema de intercomunicación suele ser un simples bus, pero tanto éste como la propia memoria se convierten rápidamente en un cuello de botella y permite un número reducido de procesadores (se dice que es poco "escalable"), que en el caso de un bus suele rondar la treintena, aunque con estructuras más sofisticadas aunque poco habituales se ha llegado a pasar de cien.

En el caso NUMA, la visión sigue siendo de memoria compartida, pero está físicamente distribuida entre los procesadores. En este caso, cada procesador tiene un "envoltorio" Hw que consigue la visión de memoria compartida, aunque al final el acceso a direcciones que no están almacenadas localmente se transforme en paso de mensajes por un sistema de intercomunicación heredado de las máquina de memoria distribuida y siempre más sofisticado que un bus, como son los hipercubos, mallas, toros, etc. Estas máquinas disfrutan de la ventaja de la "escalabildiad" de que carecían las UMAs, y llegan a alcanzar más de dos mil procesadores, como es el caso del Cray T3E.

Los clusters son sistemas de computación que unen con una red, normalmente de altas prestaciones, un conjunto de máquinas que son, cada una, de por sí autónomas o que podráin funcionar solas. Estas máquinas pueden ser desde simples PCs de bajo coso hasta, a su vez, MP como los que acabamos de describir, fundamentalmente UMA. Una capa de Sw porporciona una visión de máquina única, con todos los aspectos que aparecen en la llamada SSI, *Single System Image*.

Finalmente, en estos últimos años se ha empezado a utilizar de manera habitual como procesadores matemáticos en las máquinas para HPC la tecnología que se ha venido desarrollando para el procesamiento gráfico, GPUs, y que, al igual que los cluster, brindan la oportunidad de una gran capacidad de cálculo a un precio muy económico, si bien su programación para este propósito sigue resultando un tanto incómoda (CUDA es un buen ejemplo de un Sw diseñado con este fin). En ambos casos, y en la incorporación reciente y de manera "natural" de los procesadores *multicore*, prima sobre todo una cuestión económica o de coste, de modo que, en esencia, y salvo algunas excepciones, no se desarrolla tecnología específica para HPC sino que los sistemas que se emplean en este campo usan tecnología ya existente.

2 (1'5 puntos) Indique en qué consiste la variante de la política de coherencia de cachés mediante invalidación conocida como *Read_Broadcast* y en qué medida puede reducir el número de fallos por invalidación. Explique hasta qué punto sería útil en una situación en que existieran múltiples escritores y lectores accediendo a un mismo bloque. ¿Y en el caso de un solo escritor y varios lectores?

SOLUCIÓN

Si aplica una política de invalidación "pura", el acceso a cada una de las copias de un mismo bloque ha sido modificado por un procesador producirá un fallo (los llamados fallos por invalidación o *invalidation misses*). Con la variante *Read_Broadcast* se reduce a uno el número de este tipo de fallo por cada invalidación que se realice, puesto que se aprovecha la lectura del bloque actualizado por parte del primer procesador que se lo encuentre en estado no válido para actualizar las demás copias.

Esta variante de la política de invalidación resulta ideal en el caso de bloques en los que se acceda por un solo escritor y varios lectores, y se aleja de ese ideal conforme aumenta el número de procesadores que escriben en él, llegando al extremo donde todos los procesadores escriben en un patrón temporal de grano fino a no representar ninguna mejora con respecto a la política de invalidación pura.

3 (2 puntos) Explique el funcionamiento de la instrucción atómica `test&set .Ri, /dir`. Para máquinas de memoria compartida, emplee esta instrucción para implementar la adquisición con espera activa (*spin-lock*) de un cerrojo y de manera que reduzca en lo posible el número de escrituras. Indique cómo se comportaría la implementación propuesta en un sistema de memoria compartida con cachés privadas y en el que se utilizase

invalidación como política de coherencia.

SOLUCIÓN

La instrucción devuelve en Ri el valor previo de la dirección especificada, dir, y escribe un uno en esta posición y esto de manera atómica, i.e., con la garantía de que ningún otro procesador pueda acceder a esta misma dirección mientras dura la ejecución completa del t&s. Este comportamiento lo podemos expresar de la siguiente forma:

```
Test:  test&set  .Ri, /dir
       {
        tmp <-- /dir
        /dir <-- '1'
        Ri <-- tmp
       }
```

supuesto que las acciones comprendidas entre las llaves ocurren atómicamente.

Una primera implementación de una primitiva de adquisición de un cerrojo en la dirección lock podría ser:

```
Test:  test&set  .R1, /lock
       bnz test
       ret
```

que es una implementación con espera activa en la que en cada "vuelta" se fuerza a uno el valor del cerrojo. En la estructura que se plantea, cada procesador tendrá copia del blqoue que contiene la dirección del cerrojo y, consecuentemente, la política de invalidación hará que a cada vuelta al bucle de espera sean invalidadas las copias de los demás procesadores, con los ulteriores fallos de cache por invalidación.

Una posible manera de atenuar esta situación una estructura conocida como test&test&set, y estudiada en las clase de la asignatura, donde primero se consulta (y no se modifica) el valor del cerrojo hasta encontrarlo "abierto" (supuesta una política de coherencia el valor que se lea será el correcto, aunque produzca un fallo por invalidación) y luego tratar de ejecutar el test&set, que si que modificaría el bloque y daría lugar a las invalidaciones pertinentes. Nótese que el número de procesadores que hubiesen pasado el "filtro" de la lectura pudiera ser grande y entonces entonces en el test&set se podría estar en una situación similar a la original, aunque no tiene porque darse este caso normalmente. El pseudocódigo sería como sigue:

```
A:  while (LOAD(lock) = 1) do
nothing;
     if (TEST&SET(lock) = 0)
{
región crítica;
}
     else goto A;
```

Otra posibilidades son los reintentos con retardo o *backoff*, que experimentalmente, y en particular en el caso de un retardo progresivo exponencial, exhiben el mejor rendimiento.

4 (2 puntos) Explique el funcionamiento de la pareja de instrucciones *load_locked / store_conditional* presente en las arquitecturas RISC para construir mecanismos de sincronización en memoria compartida. Implemente la primitiva de adquisición de un cerrojo con espera activa y describa su comportamiento en cuanto al rendimiento supuesto un sistema de memoria compartida, cachés privadas y actualización como política de coherencia de cachés.

SOLUCIÓN

Esta pareja de instrucciones aparece habitualmente en las arquitecturas tipo RISC. Presupone la existencia de cachés privadas y de un mecanismo subyacente de coherencia. La idea es que una dirección a la que se acceda

en lectura con un *load_locked*, ll, se accederá luego en escritura con su instrucción recíproca, *store_conditional*, sc. La unidad de control, UC, no dejará que esta escritura sea efectiva a menos que desde que se realizó la lectura al correspondiente bloque de caché éste no se ha visto modificado o invalidado (entiéndase que sería por el mecanismo de coherencia y la actividad de otro procesador). Este funcionamiento permite implementar de manera sencilla y elegante la primitiva para adquirir un cerrojo según el funcionamiento de test&test&set mencionado en el ejercicio anterior:

```
lock: ll    .R1, /dir-cerrojo  /* ll de la dir dir_cerrojo a R1*/
      bnz   .R1, $lock        /* mientras devuelva 1, sigue leyéndolo*/
      sc   #1, /dir-cerrojo   /* intenta escribir 1 en dir */
      beqz  $lock            /* si no ha tenido éxito, vuelve a leer*/
      ret
```

donde hemos supuesto que el éxito del sc se indica con la activación del flag de cero.

En el caso de que la política de coherencia fuese actualización, la solución propuesta no deja de ser una implementación del citado test&test&set, que, independientemente de qué política se emplee, reduce el número de escrituras y, en consecuencia, el tráfico extra debido al mantenimiento de la coherencia entre las cachés.

5 (3 puntos) Sea un sistema UMA en que se emplea un mecanismo de coherencia de cachés con implementación *snoopy* y protocolo MESI. En este sistema se suceden los tres accesos siguientes y en el mismo orden en que aparecen:

a) Un procesador **i** accede en lectura a su copia local en caché de la variable A cuyo bloque correspondiente se encuentra en estado no válido (I). Suponiendo que dicho bloque está en la caché del procesador **j** en estado exclusivo (S), indique la secuencia de acciones que se da en el sistema para obtener el valor actualizado de A. Señale qué modificaciones se producen en el sistema.

SOLUCIÓN

Se tratará de un **fallo en lectura**, puesto que la copia del bloque aparece en estado I, no válido. Si el bloque está en estado S en j, podría estarlo en otros procesadores también. Supondremos que es j quien suministra la copia válida del bloque. Se darán los siguientes pasos:

1. el procesador i hace una petición a memoria a través del bus
2. la caché j (que ha detectado la petición haciendo *snooping*) pone el valor en el bus (que se arbitra)
3. se abandona el acceso a memoria
4. el procesador i copia en su caché el valor del bus y se pone en estado S
5. todas las otras copias (incluida la de j) permanecen en estado S

a) A continuación el procesador **i** escribe un nuevo valor en A. Indique la secuencia de acciones a que da lugar esta escritura y las modificaciones que conlleva en el sistema.

SOLUCIÓN

Se trata de un **acierto en escritura**, ya que el bloque se encuentra en estado S. Se darán los siguientes pasos:

1. el procesador i difunde en el bus una operación de invalidación
2. los procesadores (haciendo *snooping*) con copia en estado S (entre los que se encuentra seguro j, supuesto que el bloque no haya sido reemplazado localmente) transitan: S->I (no válido)
3. se actualiza el valor local en la caché
4. cambio en el estado local en i: S->M (modificado, única copia válida)

a) En un instante posterior, el mismo procesador **j** accede en lectura a variable A, cuyo bloque correspondiente *no* ha sido reemplazado de su caché. Indique la nueva secuencia de acciones a que da lugar esta lectura y las modificaciones a que da lugar.

SOLUCIÓN

Se trata de un nuevo **fallo en lectura**, ahora en el procesador j y con la única copia válida (estado M) en i. Se darán los siguientes pasos:

1. el procesador j hace una petición a memoria a través del bus
2. la caché i, haciendo *snooping*, pone el valor en el bus
3. se abandona el acceso a memoria
4. el procesador local j copia el valor en su caché
5. la copia local en j se pone en estado S
6. el valor origen (M) se actualiza en memoria: *copy-back*
7. el bloque origen en i cambia de estado: M -> S

1 (2 puntos) Durante una sesión de medida de media hora, un monitor software ha extraído las siguientes variables operacionales básicas de un servidor web:

Variable	Valor
A	360 peticiones
C	351 peticiones
B	23 minutos

a) ¿A qué parámetros corresponde cada una de ellas?

b) Razone si existe equilibrio de flujo de trabajos durante el intervalo de observación.

c) A partir de la información anterior calcule las siguientes variables operacionales deducidas del servidor web: tasa de llegada, productividad, utilización y tiempo medio de servicio. No olvide indicar las unidades de cada variable.

SOLUCIÓN

a) Siguiendo la notación utilizada en análisis operacional, las variables obtenidas durante el intervalo de observación corresponden a los parámetros:

A: número de llegadas

C: número de trabajos completados

B: tiempo de ocupación

b) Puesto que, durante el intervalo de observación, el sistema no es capaz de completar todos los trabajos que llegan, no existe equilibrio de flujo de trabajos.

c)

Tasa de llegada: $\lambda = \frac{A}{T} = \frac{360\ peticiones}{1800\ s} = 0,2\ peticiones/s$

Productividad: $X = \frac{C}{T} = \frac{351\ peticiones}{1800\ s} = 0,195\ peticiones/s$

Utilización: $U = \frac{B}{T} = \frac{1380\ s}{1800\ s} = 0,77\ (77\ \%)$

Tiempo medio de servicio: $S = \frac{B}{C} = \frac{1380\ s}{1351\ peticiones} = 3,93\ s/peticion$

2 (1 punto) En un sistema informático se sustituye su unidad de disco por una nueva 5 veces más rápida que la original.

a) ¿Qué fracción de mejora tiene esta unidad de disco si al hacer la sustitución se ha observado una ganancia global del rendimiento de 2?

b) ¿Sería posible que la ganancia observada hubiese sido 6?

c) Razone cuál es el valor máximo de la ganancia y con qué utilización del disco se obtendría.

SOLUCIÓN

a) Aplicando la ley de Amdahl, la fracción de tiempo en que se emplea la mejora (fracción de mejora), es:

$$f = \frac{ganancia\ mejora \times (ganancia - 1)}{ganancia \times (ganancia\ mejora - 1)} = \frac{5 \times (2-1)}{2 \times (5-1)} = 0,625\ (62,5\ \%)$$

b) Es imposible obtener una ganancia global de 6, ya que el componente mejorado lo hace en un valor menor, en concreto de 5.

c) El valor máximo de la ganancia será por tanto de 5, siempre que la unidad de disco se utilizase durante el 100 % del tiempo.

3 (3 puntos) Dada la secuencia de instrucciones que se muestra a continuación:

```
(1)   divf  f10, f0,  f2
(2)   multf f4,  f10, f8
(3)   addf  f8,  f2,  f0
(4)   subf  f10, f6,  f8
```

a) Indique, a la derecha del código, los distintos tipos de dependencias de datos existentes.

b) Indique la evolución de la ejecución de las **tres primeras** instrucciones, utilizando el algoritmo de Tomasulo, en la hoja adjunta.

c) Explique brevemente cuál es la diferencia más importante entre la ejecución de esta secuencia de instrucciones utilizando el algoritmo de Tomasulo y utilizando un pipeline con múltiples unidades funcionales que no lo utilice. ¿Se tardaría en ambos casos el mismo tiempo en ejecutar dicha secuencia de instrucciones? ¿El orden de finalización de las instrucciones sería el mismo?

SOLUCIÓN

a) Los tipos de dependencias de datos existentes son los siguientes:

RAW entre (1) y (2) y entre (3) y (4)

WAR entre (2) y (3)

WAW entre (1) y (4)

b) Véase la hoja adjunta

c) Utilizando el algoritmo de Tomasulo, la instrucción (3) pasa a ejecutarse sin tener que esperar a que se resuelva la dependencia de datos RAW entre (1) y (2). Por el contrario, si no se utilizase dicho mecanismo, la instrucción (3) no podría pasar a la etapa de decodificación, y por lo tanto no podría ejecutarse hasta que se resolviera dicha dependencia. Por todo ello, el tiempo de ejecución sería menor utilizando el algoritmo de Tomasulo, aunque el orden de finalización sería distinto, finalizando primero la instrucción (3), a continuación la (1) y finalmente la (2), mientras que sin utilizar dicho mecanismo las instrucciones finalizarían en el mismo orden en que aparecen en el código.

4 (4 puntos) Suponga el siguiente fragmento de código escrito para un computador sin pipeline.

```
(1)       add r1, r0, r0
(2)       add r4, r0, #100  ;  for (i=0; i<100; i++)
(3) for:  ld r5, #0(r10)    ;       a[i] = 2*a[i]
(4)       sll r5, r5, #1
(5)       st r5, #0(r10)
(6)       add r10, r10, #4
(7)       add r1, r1, #1
(8)       sub r3, r4, r1
(9)       bnz r3, $for      ; $ end for
(10)      add r1, r0, r0    ; for (i=0 .......)
           . . . . . .
```

Se quiere ejecutar este mismo programa en un computador con un pipeline de instrucciones de 5 etapas y tiempo de ciclo de 2 ns, que además dispone de todo tipo de mecanismos de adelantamiento (*forwarding*) y utiliza bifurcación retardada. Las tareas realizadas en cada etapa son las que se muestran a continuación:

- E1: Fetch
- E2: Decodificación, lectura de registros, evaluación de la condición y cálculo de la dirección de salto
- E3: Ejecución y cálculo de la dirección en los accesos a memoria para datos.
- E4: Lectura o escritura de datos en memoria.
- E5: Escritura en registros

a) Indique, de forma razonada, las dependencias de datos que producen parones, así como el número de ciclos de parada que debe introducir el procesador para resolverlas en cada caso.

b) Indique de forma razonada si el código se ejecutaría correctamente, y en caso contrario proponga una solución.

c) Reordene el fragmento de código de modo que se produzca el mínimo número de parones posible. Escriba el código reordenado a la derecha del original. Calcule el tiempo de ejecución del código reordenado así como el tiempo medio de ejecución por instrucción.

d) Sabiendo que el tiempo medio de ejecución por instrucción en el computador sin pipeline es de 8,5 ns, calcule la ganancia que se obtendría con el computador con pipeline, así como la productividad (*throughput*), expresada en MIPS, que se obtendría en cada caso.

SOLUCIÓN

a) Las dependencias de datos que producen parones son las siguientes:

Entre las instrucciones (3) y (4), ya que la instrucción (3) tiene disponible su resultado en la etapa 4 y la instrucción (4) lo necesita en la etapa 3.

Entre las instrucciones (8) y (9). En este caso la instrucción (8) tiene disponible su resultado en la etapa 3 y la instrucción (9) lo necesita en la etapa 2.

En ambos casos habría que introducir 1 ciclo de parada.

b) Al utilizarse bifurcación retardada y calcularse tanto la condición como la dirección de salto en la etapa 2, entraría en el pipeline, y se ejecutaría completamente la instrucción (10), que modificaría el valor de `r1`. Esto haría que el programa no se ejecutase correctamente. Una posible solución sería introducir una instrucción `NOP` detrás de la instrucción de salto.

c) Una posible solución de reordenado del código, con la que se consigue eliminar todos los ciclos de espera, además de una ejecución correcta, es la siguiente:

```
(1)             add r1, r0, r0
(2)             add r4, r0, #100
(3)    for:     ld r5, #0(r10)
(7)             add r1, r1, #1      ; Reordenada
(4)             sll r5, r5, #2
(8)             sub r3, r4, r1
(5)             st r5, #0(r10)      ; Reordenada
(9)             bnz r3, $for        ; $ end for
(6)             add r10, r10, #4    ; Reordenada
(10)            add r1, r0, r0
                . . . . . .
```

El tiempo de ejecución de este código sería:

$4 \times 2\ ns + 2 \times 2\ ns + 100 \times 7 \times 2\ ns = 1.412\ ns$

d) Partiendo del tiempo de ejecución calculado en el apartado anterior, y dado que el numero de instrucciones ejecutadas es 2+7x100 = 702 instrucciones, el tiempo medio de ejecución por instrucción en el computador con pipeline es:

$1.412\ ns/702\ instrucciones = 2\ ns/$instrucción (una instrucción por ciclo)

La ganancia obtenida sería por lo tanto de $8,5/2 = 4,25$.

La productividad, en el computador sin pipeline sería de 117,6 MIPS ($1/(8,5 \times 10^{-9})$) mientras que en el computador con pipeline sería de 500 MIPS ($1/(2 \times 10^{-9})$).

1 *El código que se muestra seguidamente es una implementación de una primitiva de adquisición de un cerrojo con espera activa* (spin-lock) *empleando la instrucción atómica* `lock.xchg .Ri, /dir`. *Indique cómo se comporta en cuanto número de fallos por invalidación* (invalidation misses) *para máquinas UMA con cachés privadas y mecanismo de coherencia por Invalidación. Reescriba la implementación propuesta de manera que se reduzca este número de fallos.*

```
          LD .R0, #D'01
intento:  lock.xchg .R0, /dir_cerrojo
          bnz $intento
          ret
```

SOLUCIÓN

Si se analiza la implementación propuesta, se observa que trata de reproducir el comportamiento de una instrucción de `test&set` mediante la instrucción atómica `lock.xchg`, tal y como se desarrolló en una de las sesiones de ejercicios de la asignatura. También presupone (dependería de la arquitectura particular que se usase) que la ejecución de la instrucción afecta a los flags aritméticos (según el valor que retorna el registro Ri, en este caso mientras no devuelva el valor cero).

En cada vuelta (*spin*) al bucle se escribe un uno en la dirección del cerrojo, tal y como lo haría un `test&set`, cuyo comportamiento se trata de emular, y dando lugar a la invalidación de bloque. Consiguientemente, se produciría un fallo por invalidación en cada una de las restantes copias del bloque, presentes en las cachés de los otros procesadores que estuviesen compitiendo por el acceso al cerrojo (que, a su vez, al realizar la escritura mediante el `lock.xchg`, darían lugar a una nueva invalidación de las demás copias).

Este comportamiento se atenuaría significativamente si primero se comprobase que el cerrojo está abierto (y sin modificar el bloque correspondiente ni dando lugar a los consiguientes invalidación y fallos), y luego se tratase de escribir un uno en él, en una estructura que se suele denominar `test&test&set` y estudiada también en la asignatura. Una posible codificación sería la siguiente reescritura del código propuesto en el enunciado:

```
test:     ld .R0, /dir_cerrojo
          cmp .R0, #D'00
          bnz $test
          ld .R0, #D'01
          lock.xchg .R0, /dir_cerrojo
          bnz $test
          ret
```

2 *Un programa ejecuta un total de* $120 \cdot 10^8$ *instrucciones. De ellas, el 75 % se ejecutan en tres ciclos de reloj y el resto lo hace en cinco ciclos. Para obtener el tiempo de ejecución de este programa se utiliza la orden* `time` *del sistema operativo, que proporciona la siguiente información:*

```
real 0m84s
user 0m34s
sys  0m1s
```

Calcule el número medio de ciclos por instrucción (CPI) en el programa, la frecuencia del procesador y la productividad expresada en MIPS.

SOLUCIÓN

Número medio de ciclos por instrucción:

$CPI = 0,75.3 + 0,25.5 = 3,5$ *ciclos/instruccion*

A partir de la información proporcionada por la orden *time*, deducimos que el tiempo de ejecución es:

$Tejecucion = user + sys = (34 + 1)s = 35s$

Sabiendo que:

$Tejecucion = Tciclo.N°instrucciones.CPI$

Calculamos la frecuencia del procesador del siguiente modo:

$frecuencia = \frac{N^{o}instrucciones.CPI}{Tejecucion} = \frac{120.10^{8}.3,5}{35} ciclos/s = 1,2GHz$

Productividad expresada en MIPS:

$MIPS = \frac{N^{o}instrucciones}{Tejecucion.10^{6}} = \frac{120.10^{8}}{35} = 342,85$

3 *Se dispone de un computador A con un procesador RISC que incorpora arquitectura Harvard en el nivel de memoria caché, y con un pipeline de instrucciones de 5 etapas:*

E1: Búsqueda de instrucción e incremento del PC.

E2: Decodificación y lectura de registros.

E3: Ejecución, cálculo de direcciones en las instrucciones `ld` *y* `st`*, comprobación de condición en los saltos condicionales y cálculo de dirección de salto.*

E4: Acceso a memoria de datos.

E5: Escritura en registros.

*El procesador no dispone de mecanismos de adelantamiento (*forwarding*), e introduce los ciclos de espera necesarios para resolver las posibles dependencias de datos y de control. Además, el ciclo de acceso al banco de registros está dividido en 2 subciclos (en el primero se escribe y en el segundo se lee).*

Se quiere sustituir este computador por otro, B, cuyo procesador tiene las mismas características que el anterior, salvo que dispone de todo tipo de mecanismos de adelantamiento e incorpora predicción dinámica de salto de 2 bits. Dicha predicción se realiza en la etapa E1 del pipeline.

Para evaluar la ganancia real que se obtendría con computador B se emplea, entre otros, el siguiente fragmento de código:

```
(1)  buc:  ld r5, 0(r10)       ; for (i=0; i<500; i++)
(2)        add r15, r5, r5     ;  for (j=0; j<100; j++)
(3)        ld r6, 0(r11)       ;    a[i][j] = 2*a[i][j]+b[i][j];
(4)        add r5, r5, r6
(5)        st r5, 0(r10)
(6)        add r1, r1, 1
(7)        add r10, r10, 4
(8)        add r11, r11, 4
(9)        sub r3, r1, r4
(10)       bnz r3, buc
(11)       add r1, r0, r0
(12)       add r2, r2, 1
(13)       sub r3, r2, r5
(14)       bnz r3, buc
```

a) *Indique razonadamente las dependencias de datos y de control que generan parones, así como el número de ciclos de espera que debe introducir cada uno de los procesadores para resolverlas. En el caso de las dependencias de control para el computador B indique el número de ciclos de espera que se introducen tanto en el caso de acierto como en el de fallo de predicción.*

b) *Calcule el tiempo de ejecución del fragmento de código en ambos casos, distinguiendo claramente entre el tiempo empleado en la ejecución de instrucciones y el correspondiente a los ciclos de parada de los distintos tipos de dependencias. Para la ejecución en el computador B suponga que la primera vez que se ejecuta cada instrucción de salto los bits de predicción tienen el valor 11.*

c) *Calcule la ganancia que se obtendría utilizando el computador B.*

d) *Cuantifique cual de las dos diferencias entre el computador A y el B influye más en el incremento de rendimiento que se consigue con B.*

SOLUCIÓN

a) A continuación se indican las instrucciones entre las que existen dependencias de datos, todas del tipo RAW, así como los ciclos de espera introducidos en cada procesador.

Instrucciones	Ciclos de espera en A	Ciclos de espera en B
(1) y (2)	2	1 (adelantamiento MEM-ALU)
(3) y (4)	2	1 (adelantamiento MEM-ALU)
(4) y (5)	2	0 (adelantamiento ALU-MEM)
(9) y (10)	2	0 (adelantamiento ALU-ALU)
(12) y (13)	2	0 " "
(13) y (14)	2	0 " "

En A las dependencias entre dos instrucciones consecutivas generan 2 ciclos de espera debido a que la instrucción que genera el dato escribe en el banco de registros en el primer subciclo de la etapa E5 y la instrucción que utiliza dicho dato lo debe leer en el segundo subciclo de la etapa E2.

En el caso de B, el uso de adelantamiento elimina los ciclos de espera en todos los casos salvo cuando la instrucción que genera el dato es una instrucción `ld`. En este caso, dicha instrucción tiene el dato disponible al finalizar la etapa E4 y la instrucción que utiliza dicho dato lo hace en la etapa E3, por lo que hay que introducir un ciclo de espera entre ambas para el funcionamiento correcto de la segunda instrucción.

En cuanto a las dependencias de control, éstas se producen debido a las dos instrucciones de salto (10) y (14). En el caso de A, introducirá 2 ciclos de espera detrás de cada instrucción de salto ejecutada, puesto que la condición y la dirección de salto se calculan en la etapa E3. En el caso de B, si hay acierto en la predicción no se introduce ningún ciclo de espera ya que la predicción se realiza en E1, mientras que en el caso de fallo se introducen 2 ciclos de espera puesto que la comprobación de la condición y por lo tanto de la predicción se realiza en E3.

b) Teniendo en cuenta los resultados del apartado anterior, el tiempo de ejecución en cada uno de los computadores es el siguiente:

Tej(A) = 4 ciclos de llenado del pipeline +

(500x100x10 + 500x4) ciclos de ejecución de instrucciones +

(500x100x8 + 500x4) ciclos de espera por dependencias RAW +

(500x100x2 + 500x2) ciclos de espera por dependencias de control =

= 4 + 502.000 + 402.000 + 101.000 = **1.005.004 ciclos**

Tej(B) = 4 ciclos de llenado del pipeline +

(500x100x10 + 500x4) ciclos de ejecución de instrucciones +

(500x100x2) ciclos de espera por dependencias RAW +

(500x2 + 2) ciclos de espera por dependencias de control =

= 4 + 502.000 + 100.000 + 1.002 = **603.006 ciclos**

En el caso de B, los 1.002 ciclos de espera por dependencias de control se deben a los 500 fallos de predicción que se producen en la ejecución de la instrucción (10), cada vez que se sale del bucle j, y al fallo de predicción de (14) cuando se sale del bucle i.

c) La ganancia que se obtendría con el computador B se calcula dividiendo el tiempo de ejecución obtenido con el computador A entre el obtenido con el computador B, que para el código del enunciado sería:

1.005.004/603.006 = **1,67**

d) Una forma de cuantificar cual de las dos diferencias entre A y B influyen más en el incremento de rendimiento que se consigue con B es la que se describe a continuación.

- Diferencia de tiempos de ejecucion entre A y B:

  1.005.004 - 603.006 = 401.998 ciclos

- Diferencia entre el numero de ciclos debidos a dependencias RAW entre A y B:

  (500x100x8 + 500x4) - (500x100x2) = 302.000 ciclos

que representa el **75,12 %** de la reduccion ((302.000/401.998)x100)

- Diferencia entre el numero de ciclos debidos a dependencias de control entre A y B:

  (500x100x2 + 500x2) - (500x2 + 2) = 99.998 ciclos

  que representa el **24,88 %** de la reduccion

Como se puede comprobar por los cálculos realizados, el mecanismo de adelantamiento tiene una mayor influencia que la predicción de salto en el incremento de rendimiento obtenido con el computador B.

4 *Sea un computador con memoria virtual paginada que utiliza dos niveles de tablas de páginas, direcciones virtuales de 36 bits y direcciones físicas de 32 bits. Para realizar la traducción utiliza también dos TLBs, una para instrucciones y otra para datos, ambas con tiempo de acceso de 2 ns, 6 entradas, totalmente asociativas y con política de reemplazo LRU. Las páginas son de 4 KB, y todas las tablas necesarias para la traducción tienen 2^{12} entradas. En este computador se está ejecutando un programa de cuyo código se ha extraído, para ser analizado, el siguiente fragmento:*

```
/*  Fragmento de código para analizar: */
max = 128 * 128;        /* = 16384 */
for (j=0; j<max; j++)
     y[j] = 2 * x[j];
verif(y);
for (j=0; j<max; j++)
     x[j] = y[j] / 3;
```

Los vectores `x` *e* `y`*, cuyos elementos ocupan un byte, comienzan a partir de las siguientes direcciones virtuales:*

`Dv(x) = 002003000H, Dv(y) = 003001000H`

a) *Indique el formato de las direcciones virtuales y describa cómo se han de interpretar las direcciones asociadas a los vectores* `x` *e* `y`*.*

b) *Suponga, además, que en un momento dado de la ejecución del código proporcionado, toda la información correspondiente a los vectores* `x` *e* `y` *se encuentra en direcciones contiguas de memoria física que comienzan en las direcciones siguientes:*

`Df(x) = 00020000H, Df(y) = 00038000H`

Suponiendo que todas las tablas de traducción se encuentran en memoria principal y que la TLB de datos no contiene información relativa a los datos de dicho fragmento de código, indique y justifique el contenido de las entradas de las distintas tablas de traducción que se utilizan para encontrar la dirección física de `x` *e* `y`*. Para ello:*

- *Tenga en cuenta que el tamaño de los vectores es el determinado por la constante* `max` *del código anterior.*
- *Asigne las direcciones arbitrarias que considere oportuno a las tablas de segundo nivel necesarias para realizar esta traducción, indicando claramente cuál es esta asignación.*

c) *Suponiendo que las variables* `i`*,* `j` *y* `max` *están asignadas a registros, y que la función* `verif` *no realiza accesos a memoria para leer o escribir datos, calcule el número de fallos que se producen en la TLB de datos al ejecutar este fragmento de código. Especifique en qué parte(s) del código se producen estos fallos. Justifique si otra política de reemplazo diferente de LRU podría mejorar los resultados en cuanto a la tasa de aciertos de esta TLB.*

d) *De todas las entradas de la TLB de datos, indique cuántas se utilizan y cuál será el contenido (etiquetas e información de traducción) de las entradas correspondientes a la primera página de* `x` *y a la última de* `y`*.*

e) *Calcule el tiempo total empleado en el acceso a los datos. Para ello suponga que los vectores* `x` *e* `y` *no pasan por la memoria cache, y que el tiempo de acceso de la memoria principal es de 40 ns. Calcule qué porcentaje de este tiempo corresponde a la traducción y cuál al acceso a los datos.*

f) *Suponga que durante la ejecución del fragmento de código a analizar se produce un cambio de contexto y se pasa a ejecutar otro programa que ejecuta un fragmento de código exactamente igual, y en el que* `x` *e* `y` *tienen asignadas además las mismas direcciones virtuales. Identifique cuántos fallos se producirán en la TLB de datos en este caso. Justifique si las direcciones físicas asignadas a* `x` *e* `y` *serán también coincidentes para los dos procesos.*

SOLUCIÓN

a) Según establece el enunciado, todas las tablas de traducción, tanto de primer nivel como de segundo, tienen 2^{12} entradas. Debido a ello, cada uno de los campos `zona` y `página` de la dirección virtual ocupa 12 bits. Los 12 bits menos significativos se utilizan para seleccionar el byte dentro de la página, ya que cada una de éstas tiene un tamaño de 4 KB.

El formato de las direcciones virtuales es el que se muestra a continuación:

Zona	Página	Byte (desplazamiento)
12 bits	12 bits	12 bits

La interpretación de las direcciones virtuales de comienzo de `x` e `y` según este formato es:

Dv(x) = `002003000H`. Separando estos dígitos hexadecimales en bloques de 12 bits equivale a:

`002 003 000 Hex` y significa Zona:2, PáginaVirtual:3, Desplazamiento:0.

Dv(y) = `00300100H`. Separando estos dígitos hexadecimales en bloques de 12 bits equivale a:

`003 001 000 Hex` y significa Zona:3, PáginaVirtual:1, Desplazamiento:0.

b) El tamaño de los vectores utilizados, `x` e `y`, es de $128 \times 128 = 16384 = 2^{14}$ *Bytes*, o, lo que es igual, 4 páginas de memoria. En consecuencia, las tablas de traducción contendrán información para traducir 4 páginas de `x` y 4 páginas de `y`.

Necesariamente se tendrá en memoria la tabla de traducción de primer nivel `TP1`, que según se ha visto en el apartado anterior, deberá indicar la dirección de la tabla 2 de segundo nivel (`TP2_2`) para `x` y de la tabla 3 de segundo nivel (`TP2_3`) para `y`. Por otra parte, en la tabla `TP2_2` estará la información necesaria para traducir cuatro páginas consecutivas a partir de la 3, para el vector `x`. En la tabla `TP2_3` se encontrará la información necesaria para traducir cuatro páginas consecutivas a partir de la 1, para el vector `y`. Asignando direcciones arbitrarias a la ubicación de las tablas `TP2_2` y `TP2_3` tal como sugiere el enunciado y teniendo en cuenta las direcciones físicas asociadas a los vectores `x` e `y`, se tendría la situación representada en la figura 1.

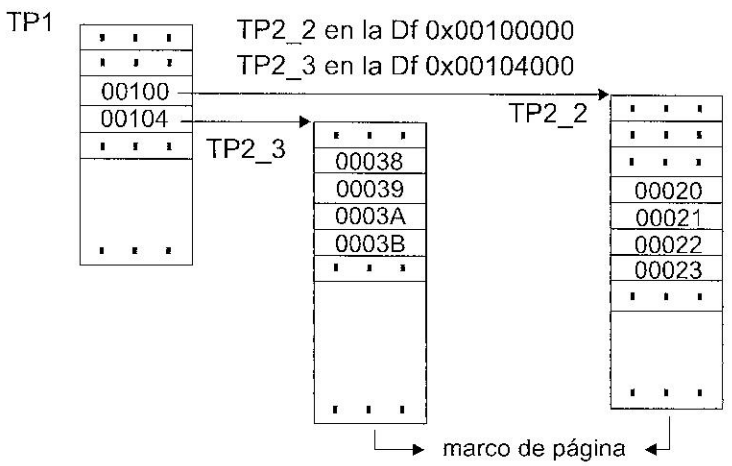

Figura 1. Contenido de las tablas de traducción.

c) Se produce un fallo de TLB la primera vez que se hace referencia a cada página. Se tendrá en cuenta que `x` e `y` ocupan cuatro páginas y que el código proporcionado realiza un acceso de lectura a `x[j]` y uno de escritura a `y[j]` en cada iteración del primer bucle.

La traza de ejecución del programa es tal que se recorren los dos vectores de forma secuencial en ambos bucles, accediendo a sus sucesivas palabras y, por lo tanto, a sus páginas consecutivas. En consecuencia, se producen los dos primeros fallos en el primer acceso a `x[0]` (zona 2, página 3) y el primero a `y[0]` (zona 3, página 1). Durante el recorrido de `x` e `y`, se irán produciendo fallos en los accesos a las páginas siguientes, llegando al final del primer bucle con 8 fallos de TLB (las cuatro páginas de cada vector) y quedando en TLB, debido a la política de reemplazo LRU, la traducción de las tres últimas páginas de `x` y las tres últimas de `y`.

Al entrar en el segundo bucle, la situación será prácticamente idéntica, ya que no se dispondrá inicialmente de la traducción de la primera página de `x` e `y`. De este modo, al actualizar la TLB se perderá la información

de traducción de la segunda página de cada vector (por LRU), lo que provocará un nuevo par de fallos de traducción, que se propagará hasta completar el segundo bucle con otro conjunto de 8 fallos.

En total se producen 16 fallos de TLB, ocho en cada uno de los bucles.

El uso de la política de reemplazo LRU hace que se produzca un nuevo fallo cada vez que se accede a una nueva página de cualquiera de los vectores utilizados en el fragmento de código de este ejemplo, por lo que otras políticas, por ejemplo, la aleatoria, podría lograr fácilmente una mejora en los resultados.

d) Tal como se desprende de lo expresado en el apartado anterior, se utilizan todas las entradas de la TLB de datos. El contenido estará formado por la etiqueta y la información de traducción. La etiqueta contendrá, al menos, la parte que se traduce de la dirección virtual. Podría contener también una identificación del proceso para el que se ha realizado la traducción. La información de traducción estará formada por la parte de la dirección física que surge de la traducción (no incluye por tanto los bits de desplazamiento dentro del marco de página), además de los permisos de acceso y otros bits para implementar las políticas de trabajo de la memoria virtual. Teniendo en cuenta únicamente la parte básica de la información mencionada, las entradas correspondientes a la primera página de x y la última de y podrían ser:

	PV	"MP ... prot. ..."
Primera página de x:	002003	00020 | ... permisos ...
Última página de y:	003004	0003B | ... permisos ...

e) Para calcular el tiempo total empleado en el acceso a datos se ha de tener en cuenta que la información utilizada no pasa por la memoria caché y las escrituras tardan el mismo tiempo que las lecturas. El número total de accesos a datos, contabilizando ambos bucles, es de:

$$2\ bucles \times 2 \times 128 \times 128 = 65.536\ accesos$$

De ellos, solo 16 se realizan con fallo de traducción en TLB (8 en cada bucle). Todos los accesos a datos, tanto con fallo como con acierto en TLB requieren un acceso a la TLB para hacer la traducción o detectar el fallo. La traducción necesaria para los 16 accesos con fallo de TLB requiere acceder a las tablas de traducción de primer y segundo nivel (es decir, dos accesos a memoria principal). Todos los accesos a datos requieren un acceso adicional a memoria para leer o escribir la información. En resumen, el tiempo invertido en los accesos a datos es:

$$t_{trad} = 65.536 \times 2\ ns + 16 \times (2 \times 40)\ ns = 132.352\ ns$$
$$t_{total} = t_{trad} + 65.536 \times 40\ ns = 2.753.792\ ns$$

En consecuencia, la proporción de tiempo dedicado a la traducción será de $132.352/2.753.792 = 0{,}048$ (4,8 %) y la dedicada a realizar los accesos a memoria para leer o escribir, será de $65.536 \times 40/2.753.792 = 0{,}952$ (95,2 %).

f) Independientemente de que la TLB tenga o no un campo identificador del proceso, se producirán también 16 fallos cada vez que se cambie de proceso, ya que se hace referencia a 8 páginas distintas en cada bucle y solo hay capacidad para traducir 6. El hecho de usar las mismas direcciones virtuales es indiferente.

Las direcciones físicas serán necesariamente distintas para los diferentes procesos ya que, al contrario que el espacio virtual, el espacio físico está compartido entre todos ellos.